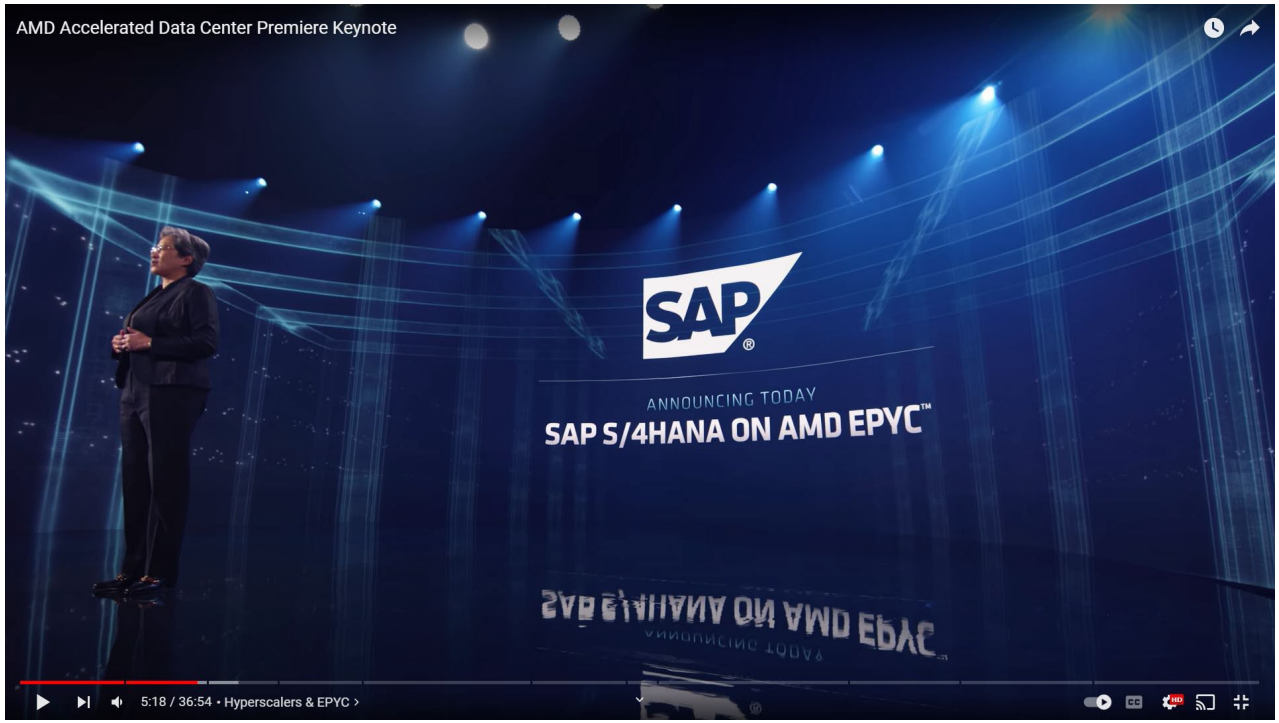
EXHIBIT 8

Exhibit 8: U.S. Patent No. 6,823,409

Claim 10	Identification
10[pre]. A coherency control module configured to control access to cache memory in a computer system, the coherency control module comprising:	To the extent the preamble is limiting, SAP provides a coherency control module configured to control access to cache memory in a computer system. For example, *see*:  Source: https://www.youtube.com/watch?v=ECHhuvuiNzs (Nov. 8, 2021)

Claim 10	Identification

AMD EPYC CPUs Now Power SAP Applications Hosted on Google Cloud

SANTA CLARA, Calif., July 11, 2023 (GLOBE NEWSWIRE) -- Today, AMD (NASDAQ: AMD) announced that SAP has chosen AMD EPYC™ processor-powered Google Cloud N2D virtual machines (VMs) to run its cloud ERP delivery operations for RISE with SAP; further increasing adoption of AMD EPYC for cloud-based workloads. As enterprises look toward digital modernization, many are adopting cloud-first architectures to complement their on-premises data centers. AMD, Google Cloud and SAP can help customers achieve their most stringent performance goals while delivering on energy efficiency, scalability and resource utilization needs.

"As part of our RISE with SAP initiative, we have made a strategic decision to add AMD EPYC processor powered N2D instances in Google Cloud to run mission critical workloads for our enterprise cloud customers." said Lalit Patil, CTO, SAP Enterprise Cloud Services, SAP SE. "Our engineering collaboration with AMD and Google Cloud can result in an increase in performance and performance-per-dollar over comparable instances."

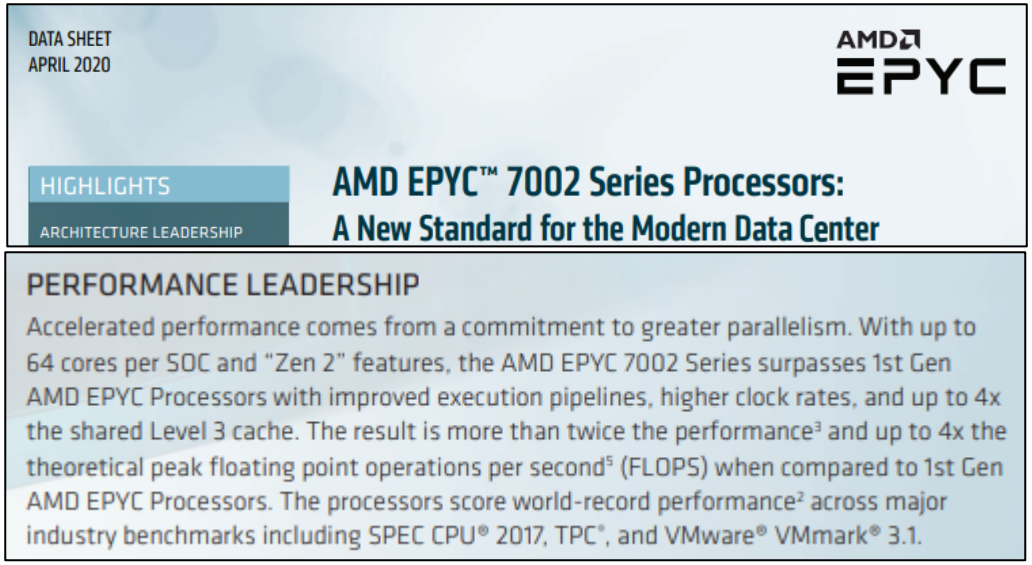
Source: https://www.amd.com/en/newsroom/press-releases/2023-7-11-amd-epyc-cpus-now-power-sap-applications-hosted-on.html

— What is the difference between SAP S/4HANA Cloud, private edition and RISE with SAP?

SAP S/4HANA Cloud, private edition is the cloud ERP at the heart of RISE with SAP. RISE with SAP bundles software that includes the cloud ERP, process intelligence and execution, advanced office of the CFO features, and more, along with a complete cloud infrastructure and migration services.

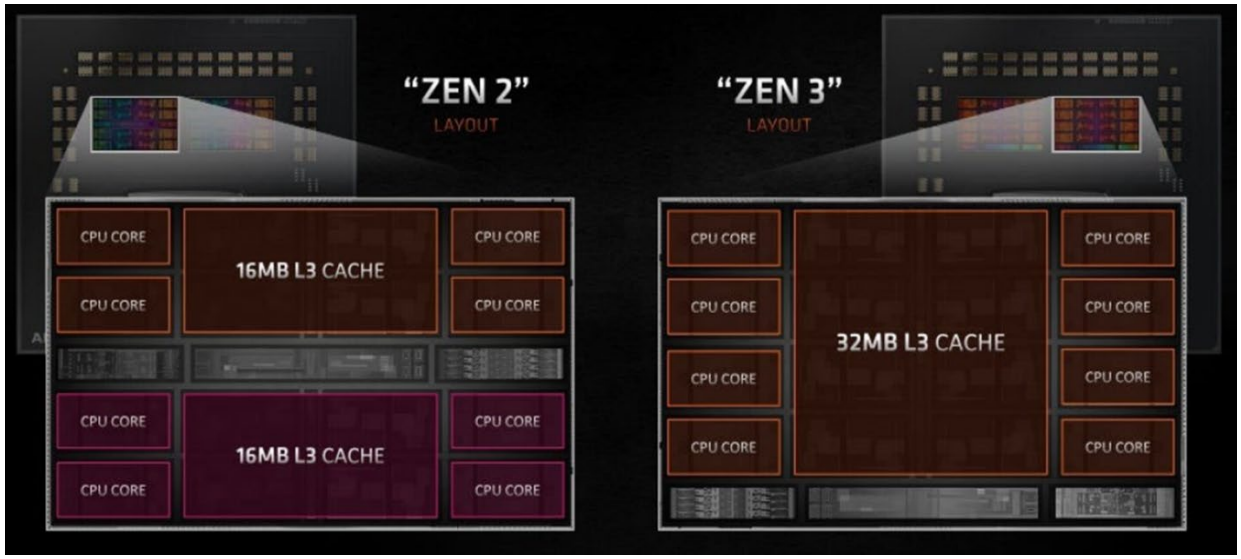
Source: https://www.sap.com/africa/products/erp/rise.html

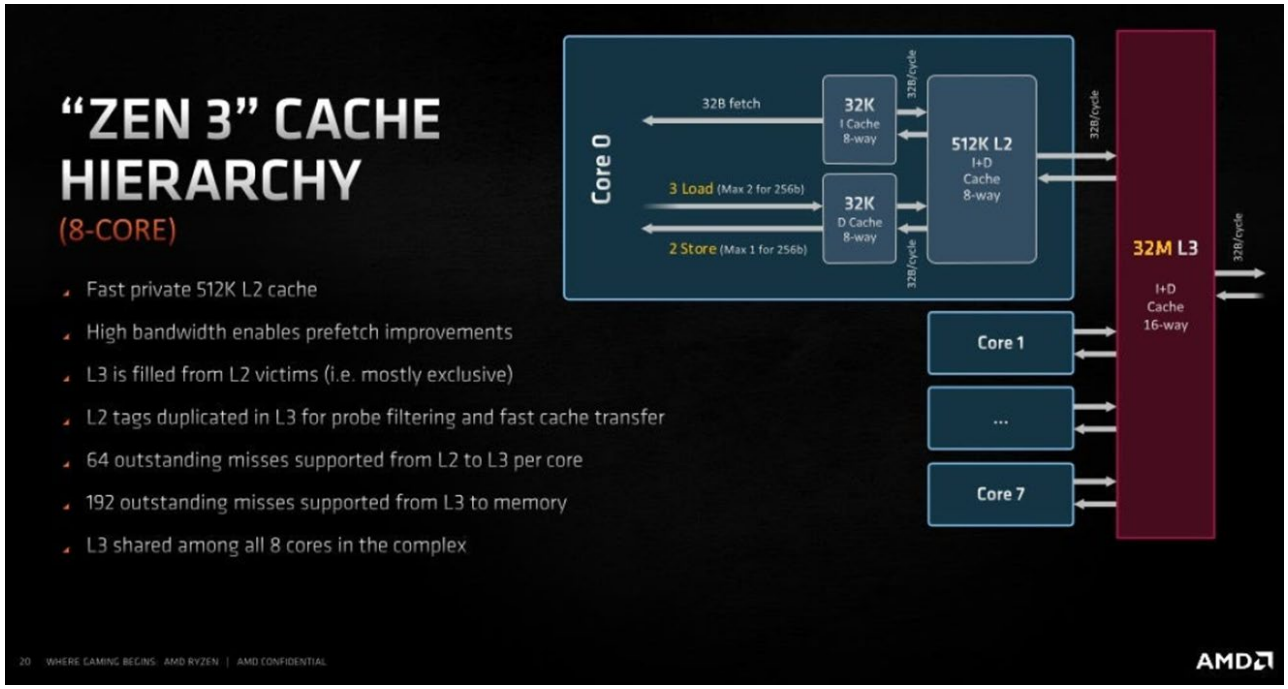
Claim 10	Identification

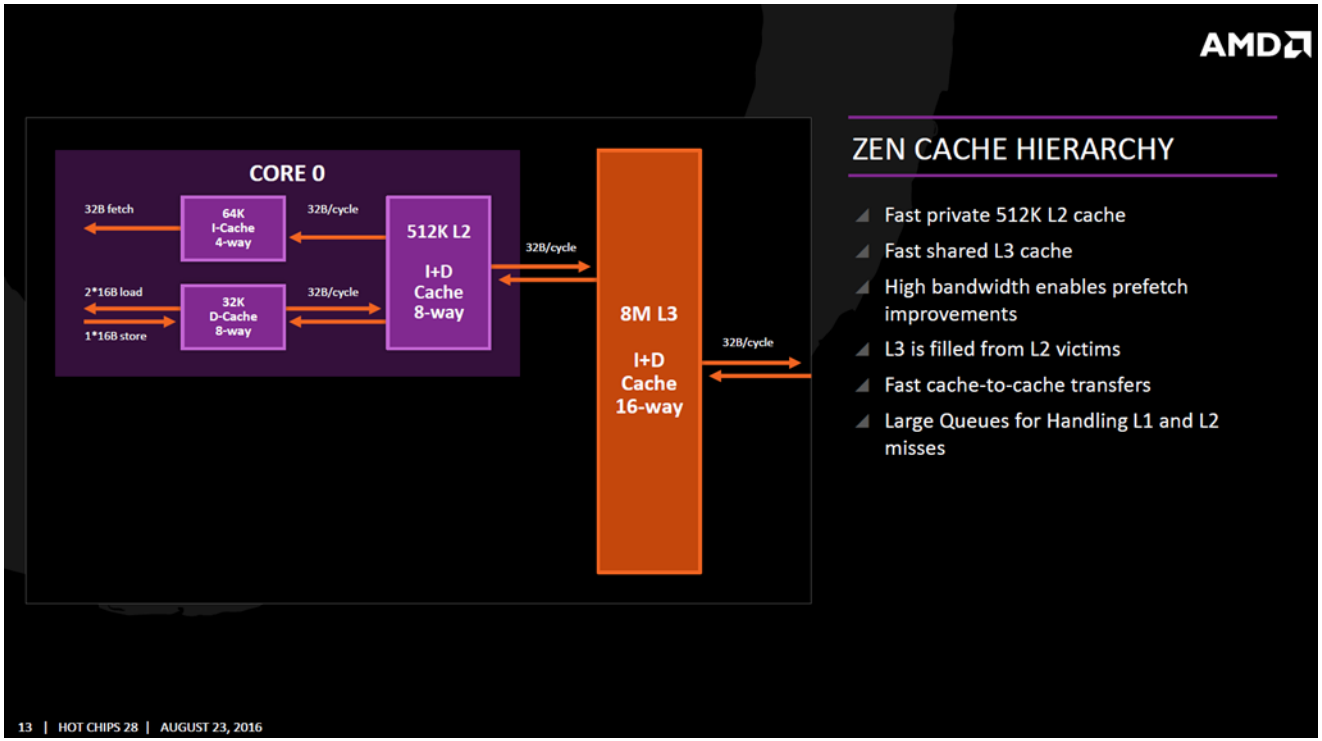


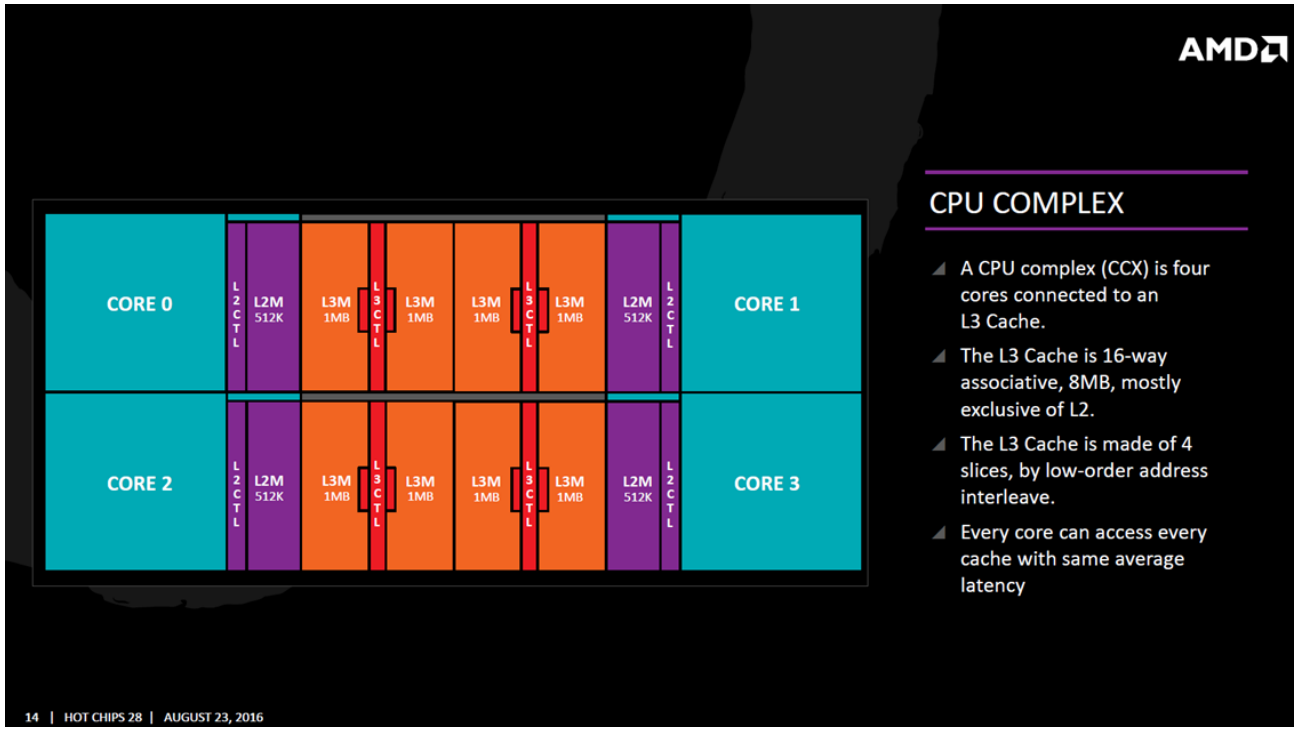
Source: https://www.amd.com/system/files/documents/AMD-EPYC-7002-Series-Datasheet.pdf; *see also* https://www.amd.com/en/processors/epyc-7002-series.

Source: https://www.amd.com/system/files/documents/amd-epyc-7003-series-datasheet.pdf; *see also* https://www.amd.com/en/processors/epyc-7003-series.

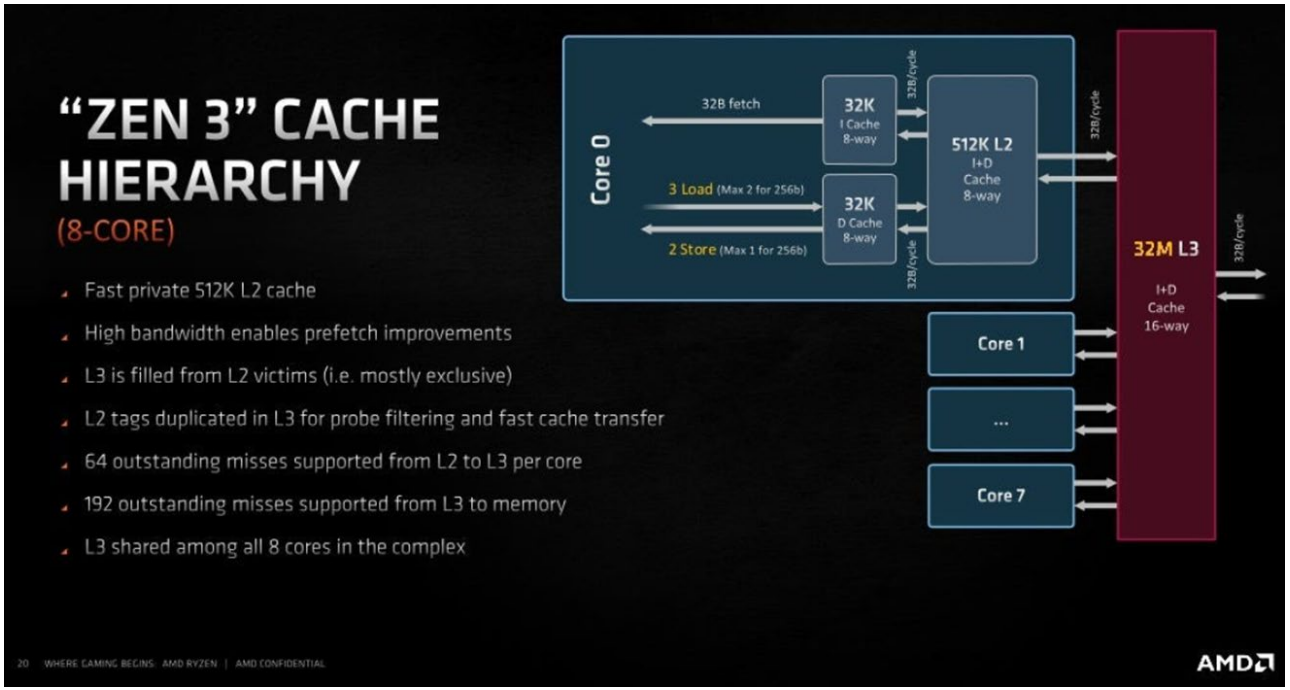
Claim 10	Identification
	 Source: https://web.archive.org/web/20220903163656/https://www.slideshare.net/slideshow/embed_code/key/6g7kfAYmt73ofd; *see also* https://web.archive.org/web/20220903163610/https://www.amd.com/en/technologies/zen-core-3.

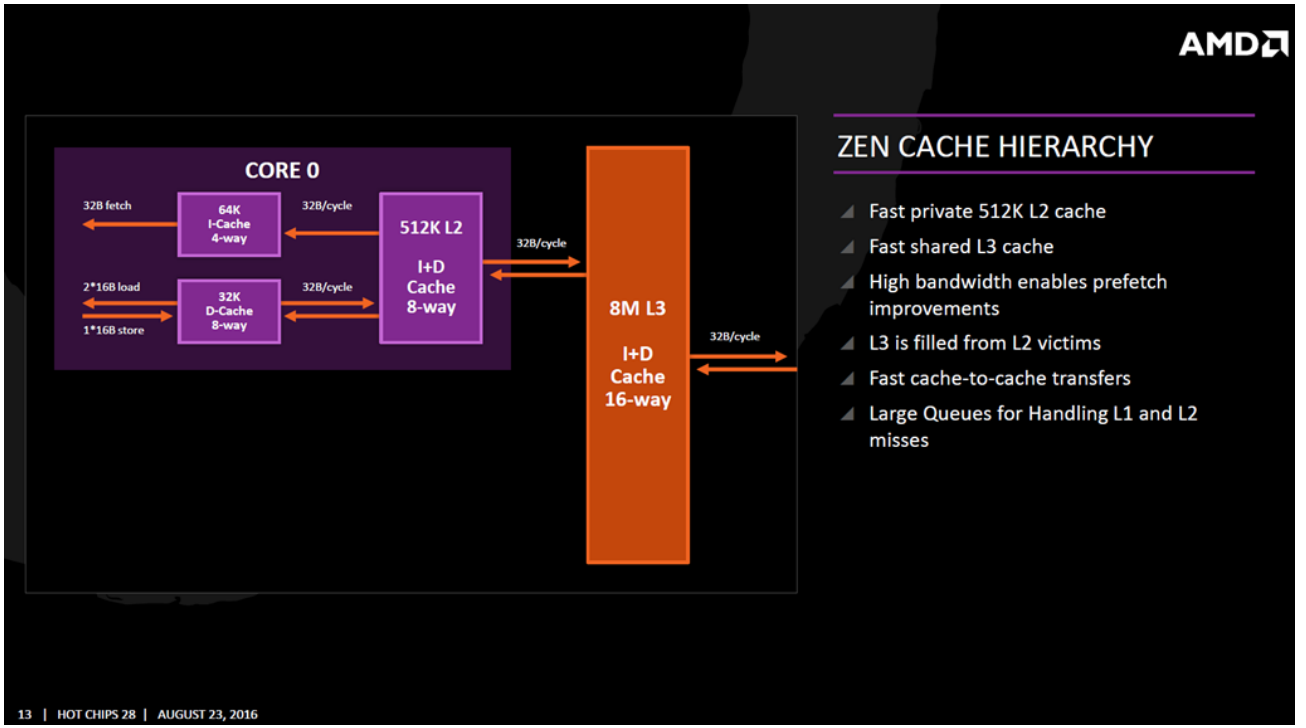
Claim 10	Identification
	 Source: https://web.archive.org/web/20220903163656/https://www.slideshare.net/slideshow/embed_code/key/6g7kfAYmt73ofd; *see also* https://web.archive.org/web/20220903163610/https://www.amd.com/en/technologies/zen-core-3.

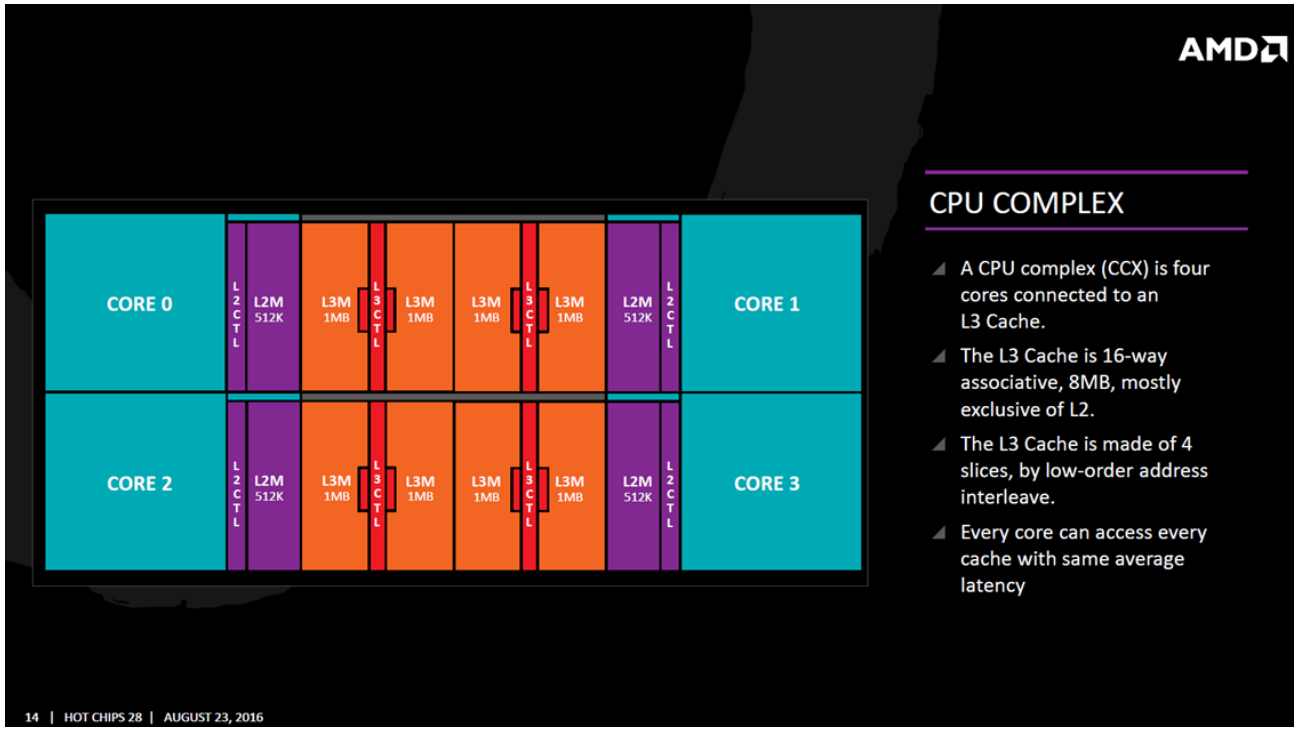
Claim 10	Identification
	 Source: https://docplayer.net/34910004-A-new-x86-core-architecture-for-the-next-generation-of-computing.html; *see* https://web.archive.org/web/20190611023103/https://www.amd.com/en/technologies/zen-core; https://web.archive.org/web/20190430213825/https://www.slideshare.net/AMD/amd-and-the-new-zen-high-performance-x86-core-at-hot-chips-28.

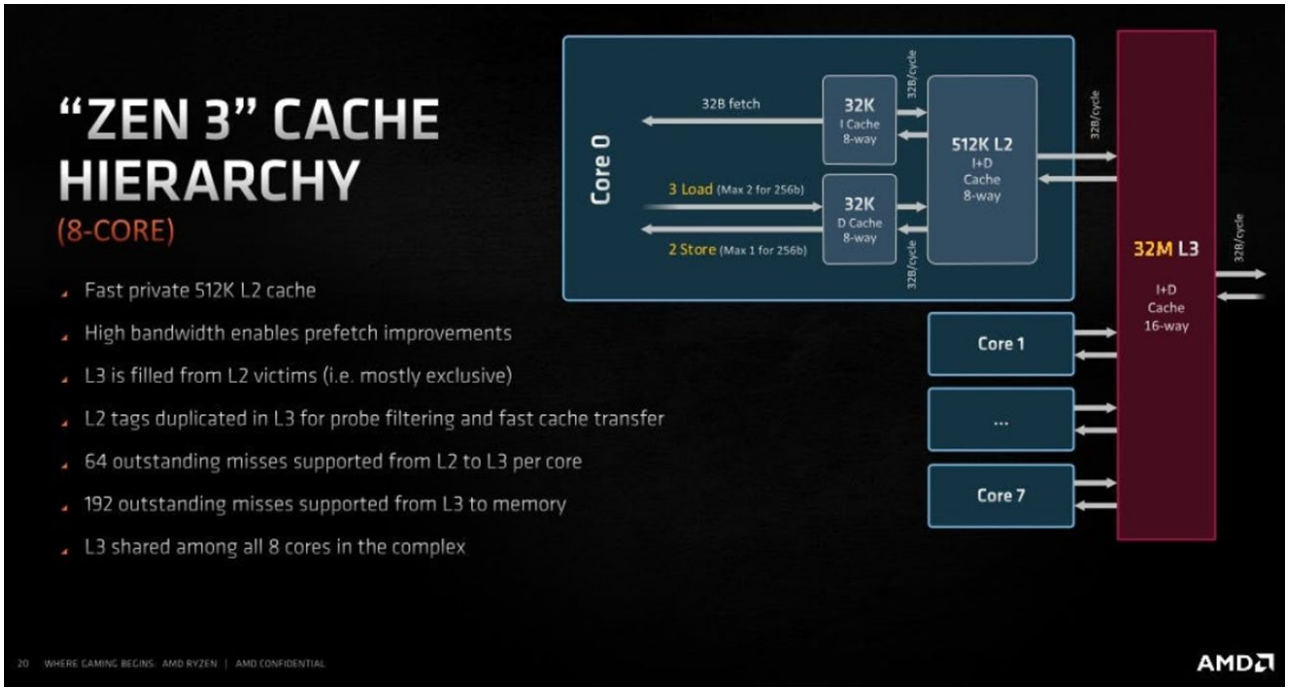
Claim 10	Identification
	 Source: https://docplayer.net/34910004-A-new-x86-core-architecture-for-the-next-generation-of-computing.html; *see* https://web.archive.org/web/20190611023103/https://www.amd.com/en/technologies/zen-core; https://web.archive.org/web/20190430213825/https://www.slideshare.net/AMD/amd-and-the-new-zen-high-performance-x86-core-at-hot-chips-28.

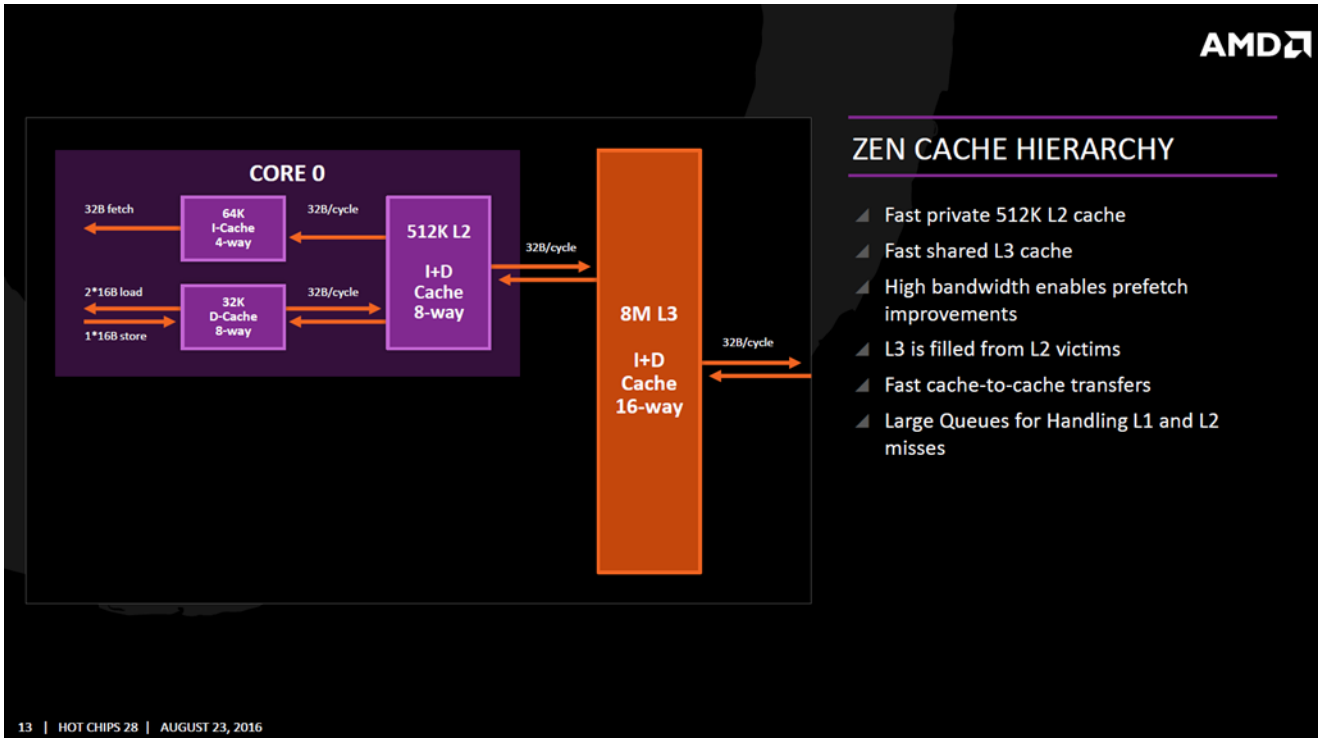
Claim 10	Identification
	7.3 Memory Coherency and Protocol Implementations that support caching support a cache-coherency protocol for maintaining coherency between main memory and the caches. The cache-coherency protocol is also used to maintain coherency between all processors in a multiprocessor system. The cache-coherency protocol supported by the AMD64 architecture is the *MOESI* (modified, owned, exclusive, shared, invalid) protocol. The states of the MOESI protocol are: • *Invalid*—A cache line in the invalid state does not hold a valid copy of the data. Valid copies of the data can be either in main memory or another processor cache. • *Exclusive*—A cache line in the exclusive state holds the most recent, correct copy of the data. The copy in main memory is also the most recent, correct copy of the data. No other processor holds a copy of the data. • *Shared*—A cache line in the shared state holds the most recent, correct copy of the data. Other processors in the system may hold copies of the data in the shared state, as well. If no other processor holds it in the *owned* state, then the copy in main memory is also the most recent. • *Modified*—A cache line in the modified state holds the most recent, correct copy of the data. The copy in main memory is stale (incorrect), and no other processor holds a copy. • *Owned*—A cache line in the owned state holds the most recent, correct copy of the data. The owned state is similar to the shared state in that other processors can hold a copy of the most recent, correct data. Unlike the shared state, however, the copy in main memory can be stale (incorrect). Only one processor can hold the data in the owned state—all other processors must hold the data in the shared state. Source: AMD64 Architecture Programmer's Manual Volume 2: System Programming (Oct. 2019) at 169 (available at https://ia803102.us.archive.org/26/items/advancedmicrodevices_24593_3.32/24593.pdf).
10[a]. a request module configured to receive requests from a plurality of buses in a computer system and to maintain proper	SAP provides a request module configured to receive requests from a plurality of buses in a computer system and to maintain proper ordering of the requests from each bus. For example, *see*:

Claim 10	Identification
ordering of the requests from each bus;	 Source: https://web.archive.org/web/20220903163656/https://www.slideshare.net/slideshow/embed_code/key/6g7kfAYmt73ofd; *see also* https://web.archive.org/web/20220903163610/https://www.amd.com/en/technologies/zen-core-3.

Claim 10	Identification
	 Source: https://docplayer.net/34910004-A-new-x86-core-architecture-for-the-next-generation-of-computing.html; *see* https://web.archive.org/web/20190611023103/https://www.amd.com/en/technologies/zen-core; https://web.archive.org/web/20190430213825/https://www.slideshare.net/AMD/amd-and-the-new-zen-high-performance-x86-core-at-hot-chips-28.

Claim 10	Identification
	 Source: https://docplayer.net/34910004-A-new-x86-core-architecture-for-the-next-generation-of-computing.html; *see* https://web.archive.org/web/20190611023103/https://www.amd.com/en/technologies/zen-core; https://web.archive.org/web/20190430213825/https://www.slideshare.net/AMD/amd-and-the-new-zen-high-performance-x86-core-at-hot-chips-28.

Claim 10	Identification
	 Source: https://web.archive.org/web/20220903163656/https://www.slideshare.net/slideshow/embed_code/key/6g7kfAYmt73ofd; *see also* https://web.archive.org/web/20220903163610/https://www.amd.com/en/technologies/zen-core-3.

Claim 10	Identification
	 Source: https://docplayer.net/34910004-A-new-x86-core-architecture-for-the-next-generation-of-computing.html; *see* https://web.archive.org/web/20190611023103/https://www.amd.com/en/technologies/zen-core; https://web.archive.org/web/20190430213825/https://www.slideshare.net/AMD/amd-and-the-new-zen-high-performance-x86-core-at-hot-chips-28.

Claim 10	Identification

7.2 Multiprocessor Memory Access Ordering

The term memory ordering refers to the sequence in which memory accesses are performed by the memory system, as observed by all processors or programs.

To improve performance of applications, AMD64 processors can speculatively execute instructions out of program order and temporarily hold out-of-order results. However, certain rules are followed with regard to normal cacheable accesses on naturally aligned boundaries to WB memory.

In the examples below, all memory values are initialized to zero.

From the point of view of a program, in ascending order of priority:

- All loads, stores and I/O operations from a single processor appear to occur in program order to the code running on that processor and all instructions appear to execute in program order.
- Successive stores from a single processor are committed to system memory and visible to other processors in program order. A store by a processor cannot be committed to memory before a read appearing earlier in the program has captured its targeted data from memory. In other words, stores from a processor cannot be reordered to occur prior to a load preceding it in program order.

  In this context:

  - Loads do not pass previous loads (loads are not reordered). Stores do not pass previous stores (stores are not reordered)

Processor 0	Processor 1
Store A ← 1	Load B
Store B ← 1	Load A

    Load A cannot read 0 when Load B reads 1. (This rule may be violated in the case of loads as part of a string operation, in which one iteration of the string reads 0 for Load A while another iteration reads 1 for Load B.)

  - Stores do not pass loads

Claim 10	Identification

Processor 0	Processor 1
Load A	Load B
Store B ← 1	Store A ← 1

Load A and Load B cannot both read 1.

- Stores from a processor appear to be committed to the memory system in program order; however, stores can be delayed arbitrarily by store buffering while the processor continues operation. Therefore, stores from a processor may not appear to be sequentially consistent.

Processor 0	Processor 1
Store A ← 1	Store B ← 1
…	…
Store A ← 2	Store B ← 2
…	…
Load B	Load A

Both Load A and Load B may read 1. Also, due to possible write combining one or both processors may not actually store a 1 at the designated location.

- Non-overlapping Loads may pass stores.

Processor 0	Processor 1
Store A ← 1	Store B ← 1
Load B	Load A

All combinations of values (00, 01, 10, and 11) may be observed by Processors 0 and 1.

- Where sequential consistency is needed (for example in Dekker's algorithm for mutual exclusion), an MFENCE instruction should be used between the store and the subsequent load, or a locked access, such as XCHG, should be used for the store.

Processor 0	Processor 1
Store A ← 1	Store B ← 1
MFENCE	MFENCE
Load B	Load A

Load A and Load B cannot both read 0.

Source: AMD64 Architecture Programmer's Manual Volume 2: System Programming (Oct. 2019) at 166-67 (available at https://ia803102.us.archive.org/26/items/advancedmicrodevices_24593_3.32/24593.pdf).

Claim 10	Identification

Table 7-3. Memory Access Ordering Rules

First Memory Operation	Second Memory Operation: Load (wp, wt, wb)	Load (uc)	Load (wc, wc+)	Store (wp, wt, wb)	Store (uc)	Store (wc, wc+, non-temporal)	Load/Store (io)	Lock (atomic)	Serialize instructions/ Interrupts/Exceptions
Load (wp, wt, wb)	a	f	b (lf)	c	c	c	d	d	d
Load (uc)	a	f	b (lf)	c	c	c	d	d	d
Load (wc, wc+)	a	f	b (lf)	c	c	c	d	d	d
Store (wp, wt, wb)	e (mf)	f	e (mf)	g	g	h (sf)	d	d	d
Store (uc)	i	f	i	g	g	h (sf)	d	d	d
Store (wc, wc+, non-temporal)	e (mf)	f	e (mf)	j (sf)	g, m	h (sf)	d	d	d
Load/Store (io)	k	k	k	k	k	l	d, k	d, k	d, k
Lock (atomic)	k	k	k	k	k	k	d, k	d, k	d, k
Serialize instruction/ Interrupts/Exceptions	l	l	l	l	l	l	d, l	d, l	d, l

a — A load (wp, wt, wb) may not pass a previous load (wp, wt, wb, wc, wc+, uc).

b — A load (wc, wc+) may pass a previous load (wp, wt, wb, wc, wc+). To ensure memory order, an LFENCE instruction must be inserted between the two loads.

c — A store (wp, wt, wb, uc, wc, wc+, nt) may not pass a previous load (wp, wt, wb, uc, wc, wc+, nt).

d — All previous loads and stores complete to memory or I/O space before a memory access for an I/O, locked or serializing instruction is issued.

e — A load (wp, wt, wb, wc, wc+) may pass a previous non-conflicting store (wp, wt, wb, wc, wc+, nt). To ensure memory order, an MFENCE instruction must be inserted between the store and the load.

f — A load or store (uc) does not pass a previous load or store (wp, wt, wb, uc, wc, wc+, nt).

g — A store (wp, wt, wb, uc) does not pass a previous store (wp, wt, wb, uc).

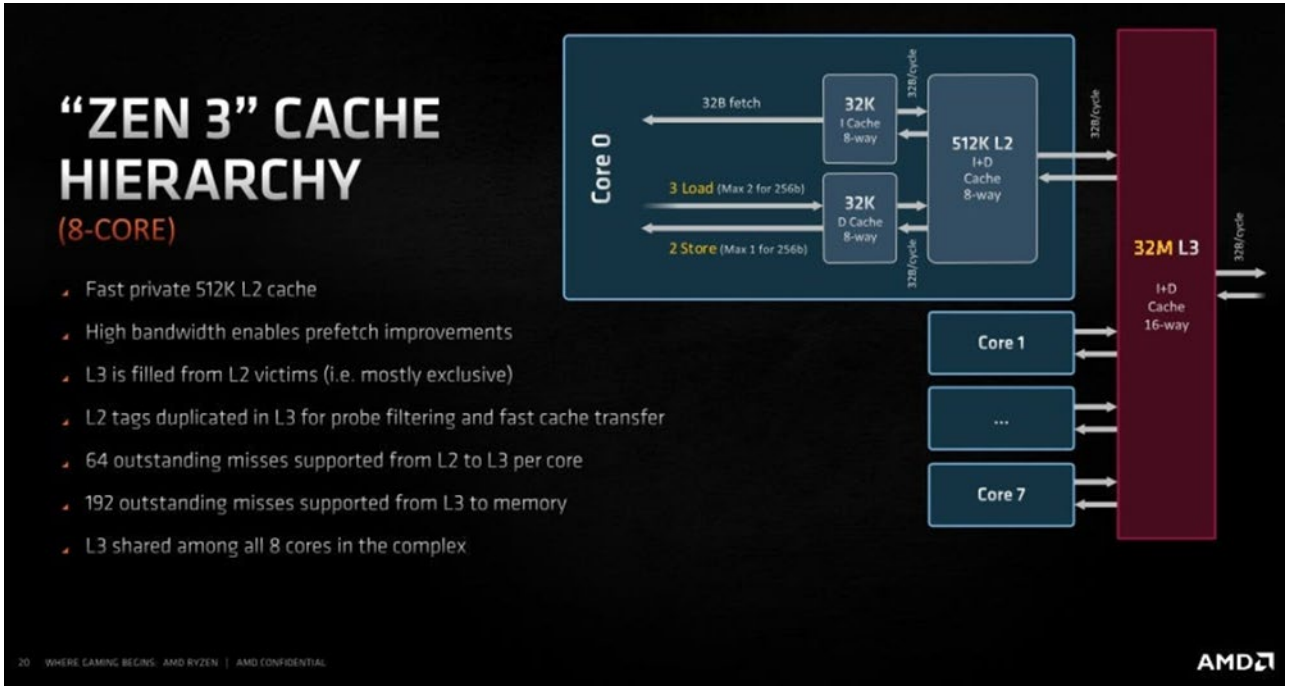
h — A store (wc, wc+, nt) may pass a previous store (wp, wt, wb) or non-conflicting store (wc, wc+, nt). To ensure memory order, an SFENCE instruction must be inserted between these two stores. A store (wc, wc+, nt) does not pass a previous conflicting store (wc, wc+, nt).

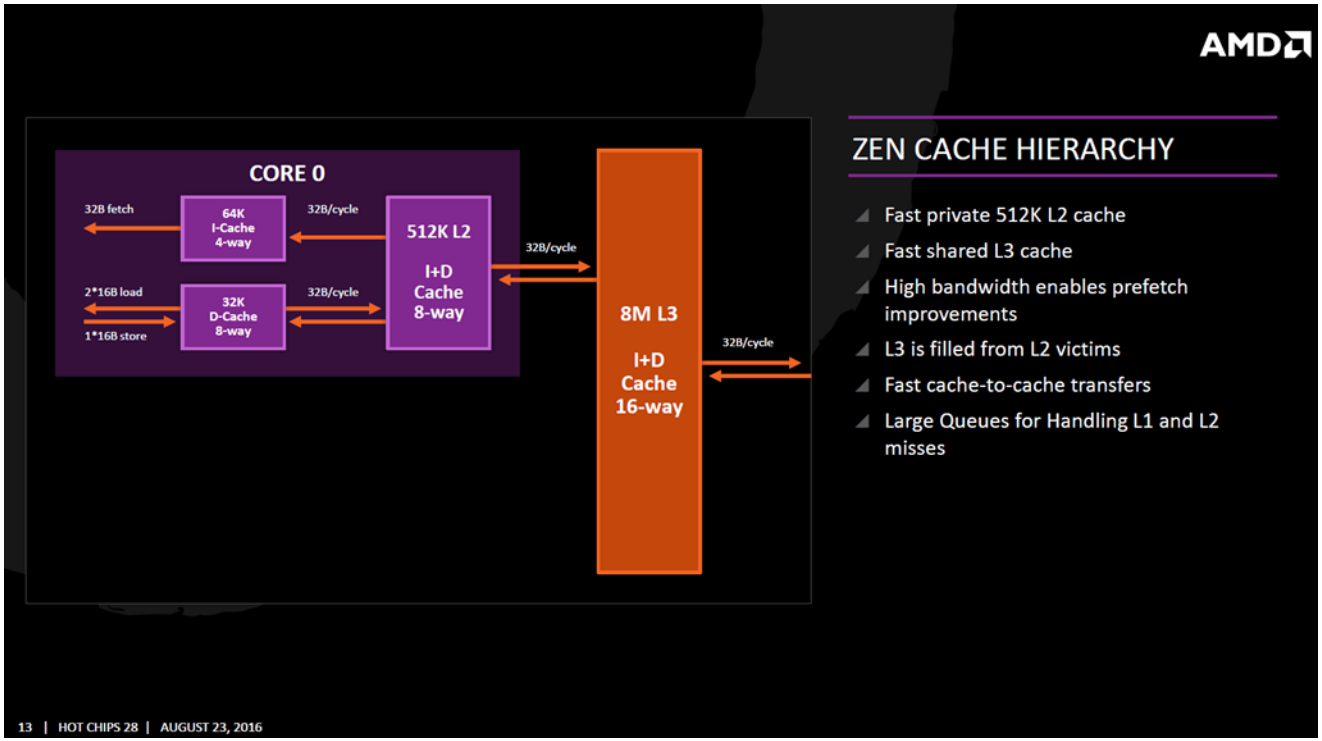
i — A load (wp, wt, wb, wc, wc+) may pass a previous non-conflicting store (uc). To ensure memory order, an MFENCE instruction must be inserted between the store and the load.

j — A store (wp, wt, wb) may pass a previous store (wc, wc+, nt). To ensure memory order, an SFENCE instruction must be inserted between these two stores.

k — All loads and stores associated with the I/O and locked instructions complete to memory (no buffered stores) before a load or store from a subsequent instruction is issued.

Source: AMD64 Architecture Programmer's Manual Volume 2: System Programming (Oct. 2019) at 176 (available at https://ia803102.us.archive.org/26/items/advancedmicrodevices_24593_3.32/24593.pdf).

Claim 10	Identification
10[b]. an active snoop queue (ASQ) module coupled to the request module and configured to maintain a list of requests from all of the buses currently being processed and to prevent multiple accesses to a single address in the cache memory simultaneously; and	SAP provides an active snoop queue (ASQ) module coupled to the request module and configured to maintain a list of requests from all of the buses currently being processed and to prevent multiple accesses to a single address in the cache memory simultaneously. For example, *see*:  Source: https://web.archive.org/web/20220903163656/https://www.slideshare.net/slideshow/embed_code/key/6g7kfAYmt73ofd; *see also* https://web.archive.org/web/20220903163610/https://www.amd.com/en/technologies/zen-core-3.

Claim 10	Identification
	 Source: https://docplayer.net/34910004-A-new-x86-core-architecture-for-the-next-generation-of-computing.html; *see* https://web.archive.org/web/20190611023103/https://www.amd.com/en/technologies/zen-core; https://web.archive.org/web/20190430213825/https://www.slideshare.net/AMD/amd-and-the-new-zen-high-performance-x86-core-at-hot-chips-28.

Claim 10	Identification
	L3 Cache Consuming the largest portion of the complex at 16 mmÂ², the level 3 cache is shared by all the cores. The L3 consists of four slices of 2 MiB per core, interleaved by the low order address. The L3 is filled with L2 victims. There are also special shadow tags found in each slice which duplicate the L2 state/tag entries for indexes in that slice. On an L2 miss or an external CCX probe in the case of multiple CCX configurations, those shadow tags are checked in parallel to the L3 checks in order to alleviate actual L2 bandwidth. The CCX itself was designed such that the L3 acts as a crossbar for each of the four cores. Source: https://fuse.wikichip.org/news/1177/amds-zen-cpu-complex-cache-and-smu/2/

Claim 10	Identification

7.2 Multiprocessor Memory Access Ordering

The term memory ordering refers to the sequence in which memory accesses are performed by the memory system, as observed by all processors or programs.

To improve performance of applications, AMD64 processors can speculatively execute instructions out of program order and temporarily hold out-of-order results. However, certain rules are followed with regard to normal cacheable accesses on naturally aligned boundaries to WB memory.

In the examples below, all memory values are initialized to zero.

From the point of view of a program, in ascending order of priority:

- All loads, stores and I/O operations from a single processor appear to occur in program order to the code running on that processor and all instructions appear to execute in program order.
- Successive stores from a single processor are committed to system memory and visible to other processors in program order. A store by a processor cannot be committed to memory before a read appearing earlier in the program has captured its targeted data from memory. In other words, stores from a processor cannot be reordered to occur prior to a load preceding it in program order.

  In this context:

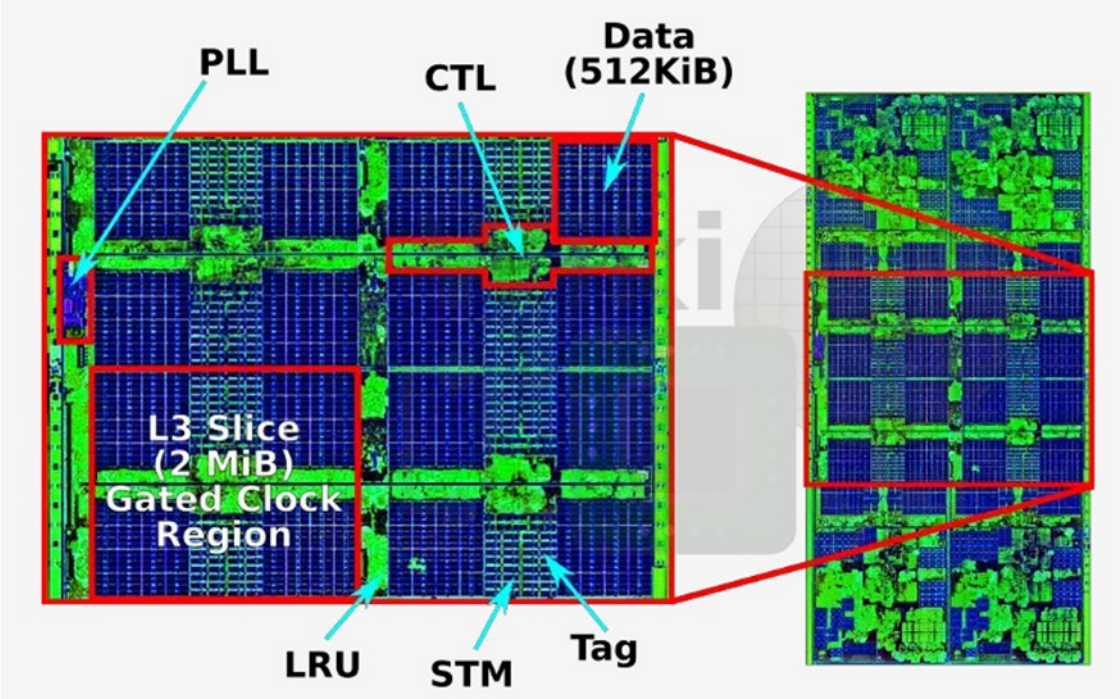
  - Loads do not pass previous loads (loads are not reordered). Stores do not pass previous stores (stores are not reordered)

Processor 0	Processor 1
Store A ← 1	Load B
Store B ← 1	Load A

    Load A cannot read 0 when Load B reads 1. (This rule may be violated in the case of loads as part of a string operation, in which one iteration of the string reads 0 for Load A while another iteration reads 1 for Load B.)

  - Stores do not pass loads

Source: AMD64 Architecture Programmer's Manual Volume 2: System Programming (Oct. 2019) at 166 (available at https://ia803102.us.archive.org/26/items/advancedmicrodevices_24593_3.32/24593.pdf).

Claim 10	Identification
10[c]. a static RAM interface module configured to access an address look-up table corresponding to data stored in the cache memory.	SAP provides a static RAM interface module configured to access an address look-up table corresponding to data stored in the cache memory. For example, *see*:  Source: https://fuse.wikichip.org/news/1177/amds-zen-cpu-complex-cache-and-smu/2/.

Claim 10	Identification

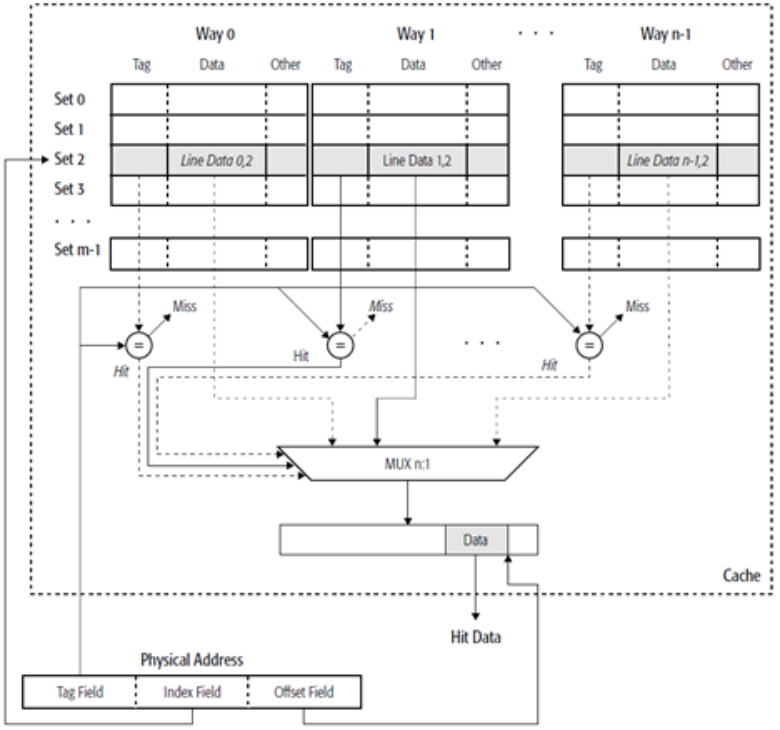


Figure 7-3. Cache Organization Example

As shown in Figure 7-3, the cache is organized as an array of cache lines. Each cache line consists of three parts: a cache-data line (a fixed-size copy of a memory block), a tag, and other information. Rows of cache lines in the cache array are *sets*, and columns of cache lines are *ways*. In an n-way set-associative cache, each set is a collection of n lines. For example, in a four-way set-associative cache, each set is a collection of four cache lines, one from each way.

Claim 10	Identification
	The cache is accessed using the physical address of the data or instruction being referenced. To access data within a cache line, the physical address is used to select the set, way, and byte from the cache. This is accomplished by dividing the physical address into the following three fields: • *Index*—The *index field* selects the cache set (row) to be examined for a hit. All cache lines within the set (one from each way) are selected by the index field. • *Tag*—The *tag field* is used to select a specific cache line from the cache set. The physical-address tag field is compared with each cache-line tag in the set. If a match is found, a cache hit is signalled, and the appropriate cache line is selected from the set. If a match is not found, a cache miss is signalled. • *Offset*—The *offset field* points to the first byte in the cache line corresponding to the memory reference. The referenced data or instruction value is read from (or written to, in the case of memory writes) the selected cache line starting at the location selected by the offset field. In Figure 7-3 on page 180, the physical-address index field is shown selecting Set 2 from the cache. The tag entry for each cache line in the set is compared with the physical-address tag field. The tag entry for Way 1 matches the physical-address tag field, so the cache-line data for Set 2, Way 1 is selected using the n:1 multiplexor. Finally, the physical-address offset field is used to point to the first byte of the referenced data (or instruction) in the selected cache line. Source: AMD64 Architecture Programmer's Manual Volume 2: System Programming (Oct. 2019) at 180-81 (available at https://ia803102.us.archive.org/26/items/advancedmicrodevices_24593_3.32/24593.pdf).